Google

Girl

Hackathon

Question And Answer

2023

Created by Anamika Kumari

Google Girl Hackathon 2023

Question 1 :

You are given a string S of length N. It consists of only two characters:
1. (: Opening bracket 0
2.): Closing bracket
S is a balanced bracket sequence. Formally, a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

- **The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.**

You are required to insert the character '+' in S such that there exists at least one '+' enclosed within each pair of matched bracket sequence. Find the minimum number of '+' characters to be inserted in S to satisfy the provided condition.

Example 1:

Input : ()(())
Output: 2

Example 2:

Input: ()((()()))
Output: 3

Constraints:

1<=T<=10
1<=N<=10^5
N is even.

Solution:

1. Intuition:

We uses a stack to keep track of opening parentheses. When a closing parenthesis is encountered, it checks for a matching opening parenthesis in the stack, indicating a valid parentheses pair. By counting the number of valid pairs, the function determines the number of valid parentheses expressions in the given string.

2. Approach:

1. We are creating a function that takes two parameters :
    a. An Integre N,
    b. A string S.

The function aims to determine the number of valid parentheses expressions in the given string.

2. Create an empty stack named st to store opening parentheses.

3. Initialize ans and flag variables to 0. ans will hold the final count of valid parenthesis expressions, and f will be used as a flag to identify the first opening parenthesis encountered.

4. Iterate through each character in the string S using a loop.

5. If the current character is an opening parenthesis (i.e., '('), push it onto the stack st. Additionally, if flag is 0 (indicating the first opening parenthesis encountered), increment ans by 1 and set flag to 1.

6. If the current character is a closing parenthesis (i.e., ')'), set flag back to 0 and pop an opening parenthesis from the stack st. This indicates a valid parentheses pair.

7. After iterating through all the characters in S, the function returns the final value of ans, which represents the count of valid parentheses expressions.

3. Time Complexity: O(N)

The time complexity of the provided code is O(N), where N is the length of the string S. The code iterates through each character in S once.

4. Space Complexity: O(N)

The space complexity is also O(N), as the stack st can potentially hold all the opening parentheses in the worst case scenario.

```
 1
 2                                        JAVA SOLUTION
 3  import java.util.Stack;
 4
 5  class Solution
 6  {
 7      public int ValidParenthesis(int N, String S)
 8      {
 9          Stack<Character> st = new Stack<>(); // to store opening
              parenthesis.
10          int ans = 0; // hold the final count of valid parenthesis
11          int flag = 0; // identify the first opening parenthesis encountered
12
13          for (int i = 0; i < N; i++)
14          {
15              // if the current char is an opening parenthesis
16              if (S.charAt(i) == '(')
17              {
18                  st.push(S.charAt(i)); // push it onto the stack
19                  // if flag is 0, set it to 1 and increment ans
20                  if (flag == 0)
21                  {
22                      ans++;
23                      flag = 1;
24                  }
25              }
26              // else set the flag back to 0 and pop the opening parenthesis
                  from the stack
27              else
28              {
29                  flag = 0;
30                  st.pop();
31              }
32          }
33          return ans;
34          // TC: O(N)
35          // SC: O(N)
36      }
37  }
38
```

```
1
2                                       C++ SOLUTION
3   class Solution
4   {
5       public:
6       int ValidParenthesis(int N, string S)
7       {
8           stack<char> st; //to store opening parenthesis.
9           int ans = 0; //hold final count of valid parenthesis
10          int flag = 0; //identify the fisrt opening parentheis encountered.
11
12          for (int i = 0; i < N; i++)
13          {
14              //if current char is a opening parenthesis
15              if (S[i] == '(')
16              {
17                  st.push(S[i]); //push it onto the stack
18                                 //if flag is 0 set it to 1 and increment
                       ans.
19                  if (flag == 0)
20                  {
21                      ans++;
22                      flag = 1;
23                  }
24              }
25              //else set flag back to 0 and pop opening parenthesis from
                  stack.
26              else
27              {
28                  flag = 0;
29                  st.pop();
30              }
31          }
32          return ans;
33          //TC: O(N)
34          //SC: O(N)
35      }
36  };
```

```
 1
 2                                       PYTHON SOLUTION
 3  class Solution :
 4      def ValidParenthesis(self, N, S):
 5          st = []  # to store opening parenthesis
 6          ans = 0  # hold the final count of valid parenthesis
 7          flag = 0  # identify the first opening parenthesis encountered
 8
 9          for i in range(N):
10              #if the current char is an opening parenthesis
11              if S[i] == '(':
12                  st.append(S[i])  # push it onto the stack
13  #if flag is 0, set it to 1 and increment ans
14                  if flag == 0:
15                      ans += 1
16                      flag = 1
17  #else set the flag back to 0 and pop the opening parenthesis from the stack
18              else:
19                  flag = 0
20                  st.pop()
21
22          return ans
23  # TC: O(N)
24  # SC: O(N)
25
```

Google Girl Hackathon 2023

Question 2 : Maximize equal numbers

You are given the following:

- **An array a consisting of n elements**
- **Integer k**

For each (1≤i≤n), perform the following operation exactly one time:

Replace a; by a; +x, where x Є [-k, k] which denotes x should lie in the range of -k and k, both inclusive.

Determine the maximum length of the subsequence of array a, such that all numbers in that subsequence are equal after applying the given operation.

Notes:

A subsequence of array a is an array that can be derived from array a by removing some elements from it. (maybe none or all of them)

Assuming 1 based indexing.

Example:1

Input: a = [1, 5, 3, 1], k = 3

Output: 4

Example:2

Input: a = [3, 3, 3, 5], k = 1

Output: 4

Example:3

Input: a = [2, 5 ,1 , 2], k = 1

Output: 3

Constraints:

1<=T<=5

1<=n<=5*10^8

0<=k<=10^9

1<=a[i]<=10^9

Solution:

1. Intuition:

We will create a vector of pairs to represent the ranges, Sorting them based on the lower bounds, and counting the maximum number of overlapping intervals using a stack-like mechanism.

2. Approch:

1. For each test case, it reads the values of `n`, `k`, and `a`. `n` represents the number of elements in the array `a`, and `k` represents the range within which two numbers are considered equal.

2. It creates a vector of `Pair` objects called `v`, where each `Pair` contains the lower and upper bounds of the range for each element in `a`. The lower bound is obtained by subtracting `k` from the element, and the upper bound is obtained by adding `k` to the element.

3. It calls the `overlap` function with the `v` vector. The `overlap` function calculates the maximum number of overlapping intervals.

4. Inside the `overlap` function, it initializes variables `ans` and `count` to 0 and creates an empty vector called `data`.

5. It iterates through each `Pair` in the input vector `v` and adds two pairs to the `data` vector. One pair represents the start of the interval, denoted by the `first` value of the original pair and the character 'x'. The other pair represents the end of the interval, denoted by the `second` value of the original pair and the character 'y'.

6. The `data` vector is sorted in ascending order based on the `first` value using the `PairComparator` struct.

7. It then iterates through the sorted `data` vector. If the second value of a pair is 'x', it increments the `count` variable, indicating the start of an interval. If the second value is 'y', it decrements the `count` variable, indicating the end of an interval. The `ans` variable keeps track of the maximum `count` encountered.

8. Finally, the `overlap` function returns the maximum `ans`, which represents the maximum number of overlapping intervals.

9. The `maximumEqualNumbers` function calls the `overlap` function with the `v` vector obtained from `a` and returns the result.

10. In the `main` function, it reads the number of test cases (`tc`) from the input. Then, in a loop, it reads the values of `n`, `k`, and `a` for each test case.

11. It calls the `maximumEqualNumbers` function with the inputs and stores the result in the `result` variable.

12. Finally, it prints the `result` for each test case.

3. **Time Complexity: O(N log N)**

1. The loop in the `overlap` function that creates the `data` vector takes O(N) time, where N is the size of the input vector `v`.

2. The sorting operation using `sort` function on the `data` vector takes O(N log N) time since it compares and rearranges the elements based on the `PairComparator` struct.

3. The loop in the `overlap` function that iterates through the sorted `data` vector takes O(N) time since it processes each element once.

4. Overall, the dominant time complexity is O(N log N) due to the sorting operation.

4. **Space Complexity: O(N)**

1. The space required by the `Pair` struct is constant since it only contains two integers.

2. The `v` vector created in the `maximumEqualNumbers` function takes O(N) space since it stores N `Pair` objects.

3. The `data` vector created in the `overlap` function takes O(N) space since it contains 2N elements.

4. The stack-like structure created by the `stack` object takes O(N) space as well.

5. Overall, the space complexity of the code is O(N) due to the space usage of the vectors and stack.

Therefore, the time complexity of the code is O(N log N), and the space complexity is O(N), where N is the number of intervals or the size of the input vector.

```

JAVA SOLUTION

    import java.util.*;

// Define a Pair class to represent a pair of integers
class Pair
{
    int first;
    int second;

    Pair(int first, int second)
    {
        this.first = first;
        this.second = second;
    }
}

// Define a PairComparator class to compare pairs based on first and second values
class PairComparator implements Comparator<Pair> {
    public int compare(Pair p1, Pair p2)
{
    if (p1.first != p2.first)
        return Integer.compare(p1.first, p2.first);
    else
        return Integer.compare(p1.second, p2.second);
}
}

class Main
{
    // Function to calculate the maximum overlap
    static int overlap(List<Pair> v)
    {
        int ans = 0; // Holds the maximum overlap count
        int count = 0; // Tracks the current overlap count
        List<Pair> data = new ArrayList<>(); // Stores the intervals as pairs with labels 'x' and 'y'

        // Convert each interval into a pair of 'x' and 'y' labels
        for (int i = 0; i < v.size(); i++)
        {
            data.add(new Pair(v.get(i).first, 'x'));
            data.add(new Pair(v.get(i).second, 'y'));
        }

        // Sort the intervals in ascending order based on the values
        Collections.sort(data, new PairComparator());
```

```
48
49          // Iterate over the intervals and update the overlap count
50          for (int i = 0; i < data.size(); i++)
51          {
52              if (data.get(i).second == 'x')
53                  count++;
54              if (data.get(i).second == 'y')
55                  count--;
56              ans = Math.max(ans, count);
57          }
58
59          return ans; // Return the maximum overlap count
60      }
61
62      // Function to find the maximum equal numbers
63      static int maximumEqualNumbers(int n, int k, List<Integer> a)
64      {
65          List<Pair> v = new ArrayList<>(); // Stores the intervals
66
67          // Convert each number into an interval and store it in the list
68          for (int i = 0; i < n; i++)
69          {
70              v.add(new Pair(a.get(i) - k, a.get(i) + k));
71          }
72
73          return overlap(v); // Calculate the maximum overlap using the
              intervals
74      }
75
76      public static void main(String[] args)
77      {
78          Scanner scanner = new Scanner(System.in);
79          int tc = scanner.nextInt();
80
81          while (tc-- > 0)
82          {
83              int n = scanner.nextInt();
84              int k = scanner.nextInt();
85              List<Integer> a = new ArrayList<>();
86
87              // Read the numbers
88              for (int i = 0; i < n; ++i)
89              {
90                  a.add(scanner.nextInt());
91              }
92
93              // Calculate the maximum equal numbers and print the result
94              int result = maximumEqualNumbers(n, k, a);
95              System.out.println(result);
```

```
 96          }
 97
 98          scanner.close();
 99      }
100  }
101
102
```

```cpp

                                    C++ SOLUTION
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Define a Pair struct to represent a pair of integers
struct Pair
{
    int first;
    int second;

    Pair(int first, int second) : first(first), second(second) { }
};

// Define a PairComparator struct to compare pairs based on first and second values
struct PairComparator
{
    bool operator()(const Pair& p1, const Pair& p2) const {
        if (p1.first != p2.first)
            return p1.first<p2.first;
        else
            return p1.second<p2.second;
    }
};

// Function to calculate the maximum overlap
int overlap(vector<Pair>& v)
{
    int ans = 0;                // Holds the maximum overlap count
    int count = 0;              // Tracks the current overlap count
    vector<Pair> data;          // Stores the intervals as pairs with labels 'x' and 'y'

    // Convert each interval into a pair of 'x' and 'y' labels
    for (int i = 0; i < v.size(); i++)
    {
        data.push_back(Pair(v[i].first, 'x'));
        data.push_back(Pair(v[i].second, 'y'));
    }

    // Sort the intervals in ascending order based on the values
    sort(data.begin(), data.end(), PairComparator());

    // Iterate over the intervals and update the overlap count
    for (int i = 0; i < data.size(); i++)
    {
```

```
48          if (data[i].second == 'x')
49              count++;
50          if (data[i].second == 'y')
51              count--;
52          ans = max(ans, count);
53      }
54
55      return ans;                 // Return the maximum overlap count
56  }
57
58  // Function to find the maximum equal numbers
59  int maximumEqualNumbers(int n, int k, vector<int>& a)
60  {
61      vector<Pair> v;             // Stores the intervals
62
63      // Convert each number into an interval and store it in the vector
64      for (int i = 0; i < n; i++)
65      {
66          v.push_back(Pair(a[i] - k, a[i] + k));
67      }
68
69      return overlap(v);          // Calculate the maximum overlap using the
        intervals
70  }
71
72  int main()
73  {
74      int tc;
75      cin >> tc;
76
77      while (tc--)
78      {
79          int n, k;
80          cin >> n >> k;
81          vector<int> a(n);
82
83          // Read the numbers
84          for (int i = 0; i < n; ++i)
85          {
86              cin >> a[i];
87          }
88
89          // Calculate the maximum equal numbers and print the result
90          int result = maximumEqualNumbers(n, k, a);
91          cout << result << endl;
92      }
93
94      return 0;
95  }
```

```

                                                PYTHON SOLUTION

class Pair :
    def __init__(self, first, second):
        self.first = first
        self.second = second

class PairComparator :
    def __lt__(self, p1, p2):
        if p1.first != p2.first:
            return p1.first < p2.first
        else:
            return p1.second < p2.second

def overlap(v):
    ans = 0  # Holds the maximum overlap count
    count = 0  # Tracks the current overlap count
    data = []  # Stores the intervals as pairs with labels 'x' and 'y'

    # Convert each interval into a pair of 'x' and 'y' labels
    for i in range(len(v)):
        data.append(Pair(v[i].first, 'x'))
        data.append(Pair(v[i].second, 'y'))

    # Sort the intervals in ascending order based on the values
    data.sort(key = lambda x: (x.first, x.second))

    # Iterate over the intervals and update the overlap count
    for i in range(len(data)):
        if data[i].second == 'x':
            count += 1
        if data[i].second == 'y':
            count -= 1
        ans = max(ans, count)

    return ans  # Return the maximum overlap count

def maximumEqualNumbers(n, k, a):
    v = []  # Stores the intervals

    # Convert each number into an interval and store it in the list
    for i in range(n):
        v.append(Pair(a[i] - k, a[i] + k))

    return overlap(v)  # Calculate the maximum overlap using the intervals

tc = int(input())

```

```
while tc > 0:
    n, k = map(int, input().split())
    a = list(map(int, input().split()))

    # Calculate the maximum equal numbers and print the result
    result = maximumEqualNumbers(n, k, a)
    print(result)

    tc -= 1


```